Web Security: Session management and CSRF

CS 161: Computer Security

Ruta Jawale and Rafael Dutra

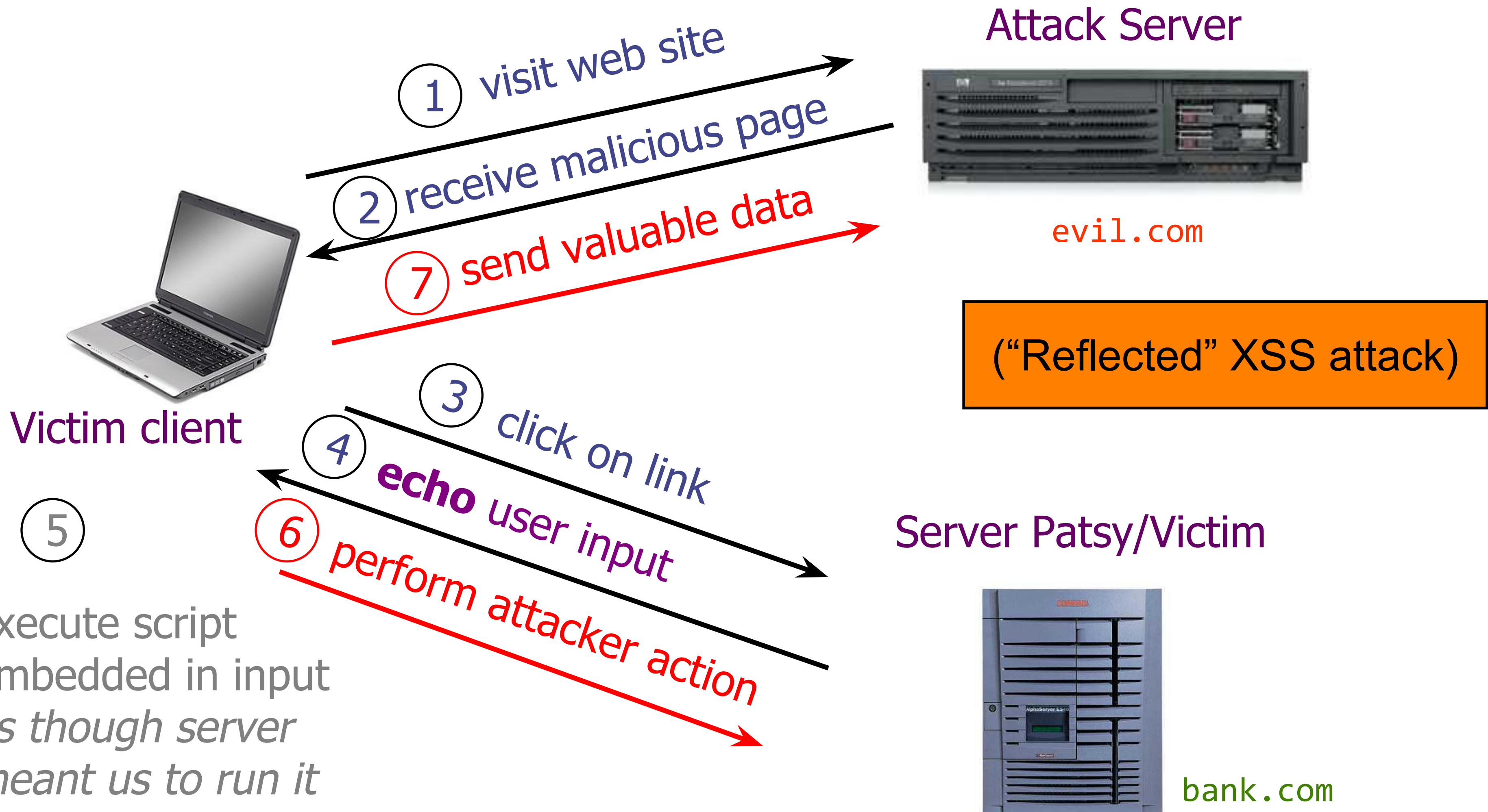
July 31, 2019

Slides credit: Raluca Ada Popa, David Wagner, Dan Boneh

Announcements

- Project 3 will be released later today
- Office Hours are moving location! (~8/1)
- Homework 2 due this Friday (8/2)
- Midterm 2 is next Monday (8/5)
  - Attend lectures and discussions

Reflected XSS (Cross-Site Scripting)



Demo

XSS Prevention

Preventing XSS

Web server must perform:

- Input validation: check that inputs are of expected form (whitelisting)
  - Avoid blacklisting; it doesn't work well
- Output escaping: escape dynamic data before inserting it into HTML

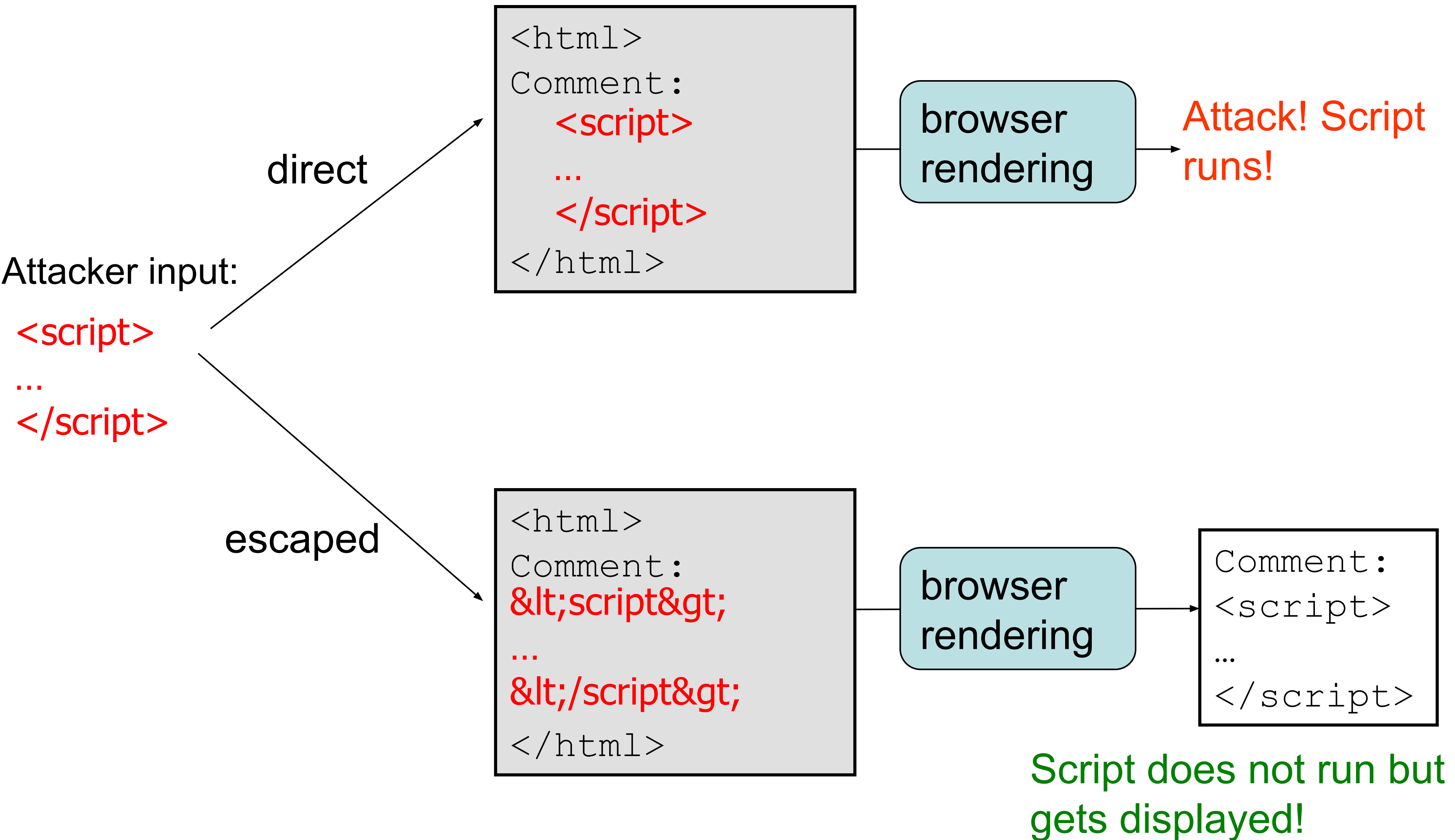
Output escaping

- HTML parser looks for special characters: < > & ” ’
  - <html>, <div>, <script>
  - such sequences trigger actions, e.g., running script
- Ideally, user-provided input string should not contain special chars
- If one wants to display these special characters in a webpage without the parser triggering action, one has to escape the parser

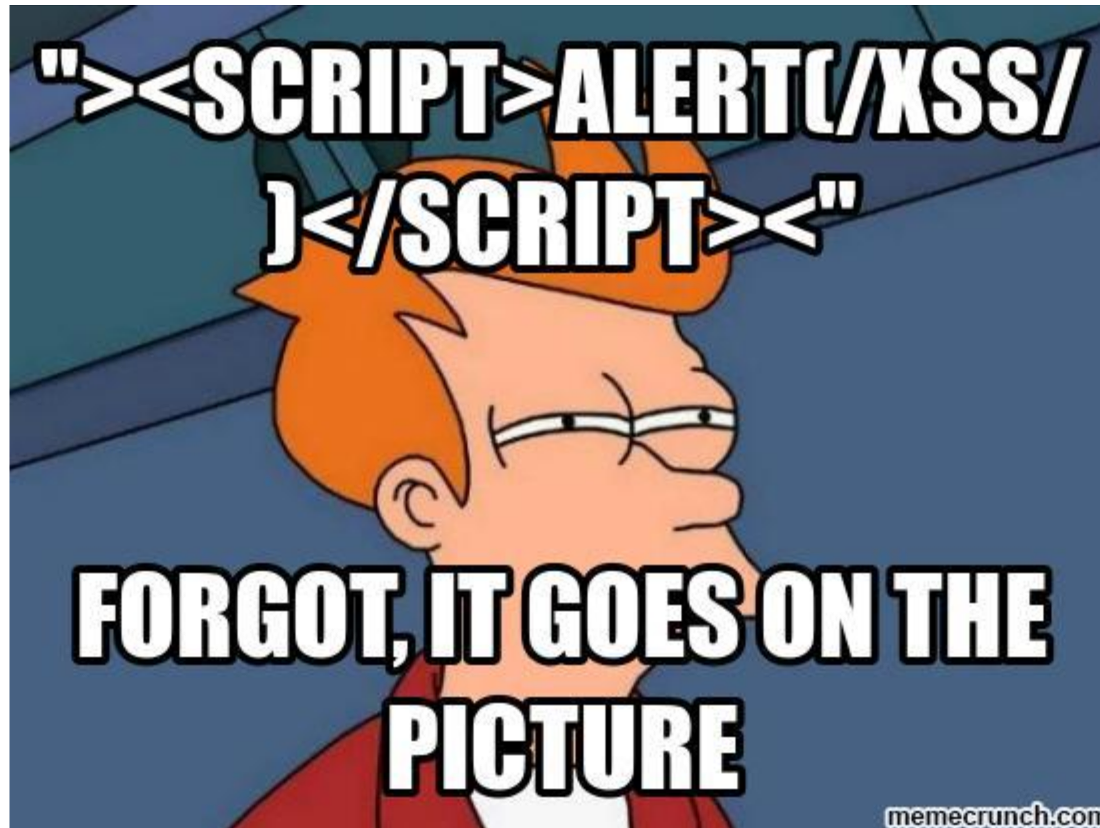
Character	Escape sequence
<	<
>	>
&	&
"	"
'	'

Demo + fix

Direct vs escaped embedding



Escape user input!



XSS prevention (cont'd): Content-security policy (CSP)

- Have web server supply a whitelist of the scripts that are allowed to appear on a page
  - Web developer specifies the domains the browser should allow for executable scripts, disallowing all other scripts (including **inline scripts**)
- Can opt to globally dis-allow script execution

HTTP Cookie

HTTP is mostly stateless

- Apps do not typically store persistent state in client browsers
  - User should be able to login from any browser
- Web application servers are generally "stateless":
  - Most web server applications maintain no information in memory from request to request
    - Information typically stored in databases
  - Each HTTP request is independent; server can't tell if 2 requests came from the same browser or user.
- Statelessness not always convenient for application developers: need to tie together a series of requests from the same user

Outrageous Chocolate Chip Cookies

1676 reviews

Made 321 times

Recipe by: Joan

"A great combination of chocolate chips, oatmeal, and peanut butter."

Save | I Made it | Rate it | Share | Print

Ingredients

25 m | 18 servings | 207 cals

- 1/2 cup butter
- 1/2 cup white sugar

Market Pantry Granulated Sugar - 4lbs
$2.59
SEE DETAILS
ADVERTISEMENT

- 1/3 cup packed brown sugar
- 1 cup all-purpose flour
- 1 teaspoon baking soda
- 1/4 teaspoon salt
- 1/2 cup rolled oats
- 1 cup semisweet chocolate chips

On Sale On

What's on sale near you.

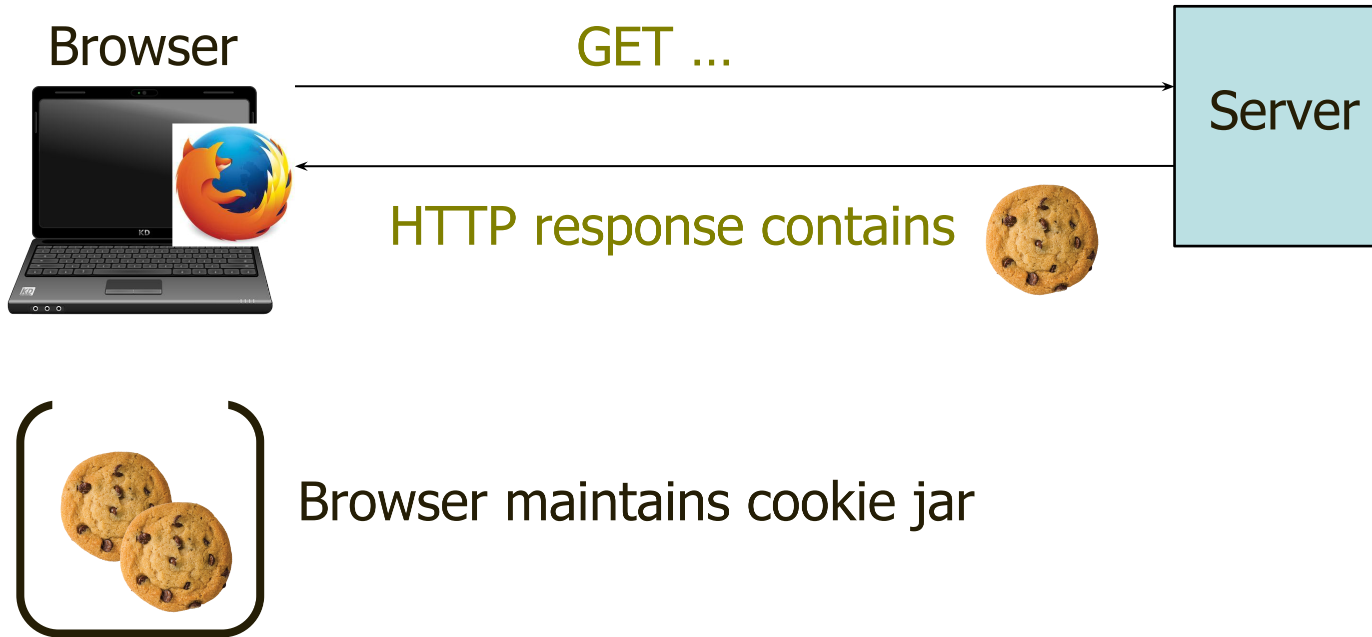
Target
1057 Eastshore Hwy
ALBANY, CA 94710
Sponsored

May we suggest

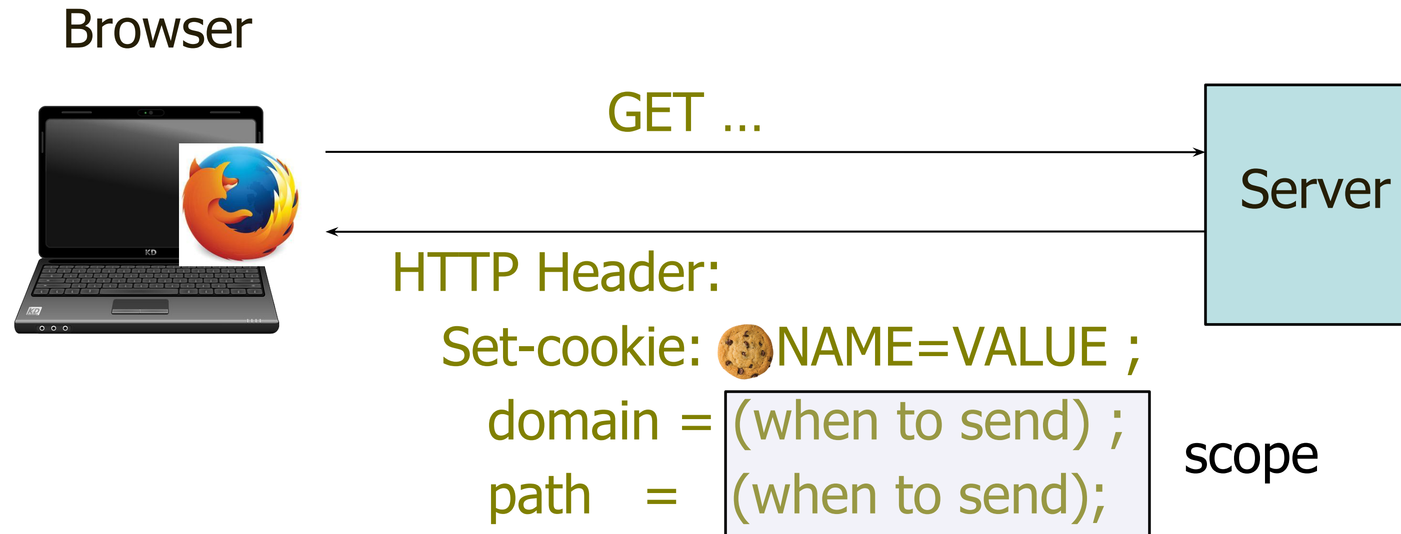
These nearby stores have ingredients on sale!

Cookie

- A way of maintaining state

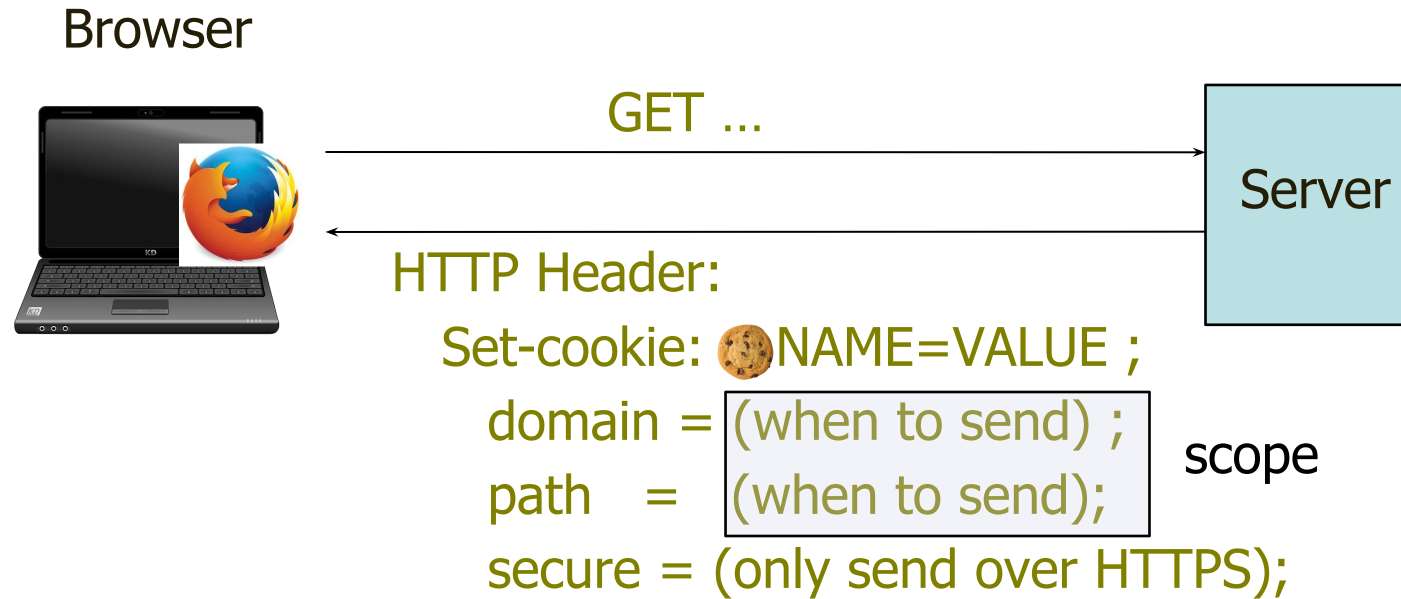


Cookie Scope



- When the browser connects to the same server later, it includes a Cookie: header containing the name and value, which the server can use to connect related requests.
- Domain and path inform the browser about which sites to send this cookie to

Secure Cookie



- **Secure** flag: cookie sent over https only
  - https provides secure communication (privacy and integrity)

HTTP-Only Cookie

- Expires is expiration date
  - Delete cookie by setting "expires" to date in past
- **HttpOnly** Flag: cookie cannot be accessed by Javascript, but only sent by browser
  - Prevents XSS, not CSRF from stealing cookies

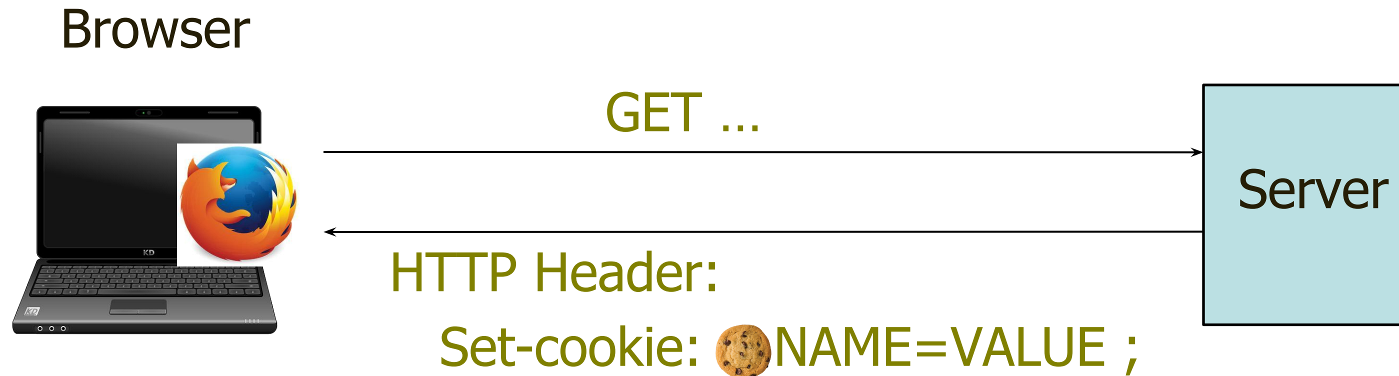
Cookie Policy

Cookie Policy

Given site A. Rules based on cookie scope:

1. Which cookies can be set?
   - Cookies with domain-suffix (aka super domain) of site A (except TLD)
2. Which cookies can be received?
   - Cookies with domain-suffix (aka super domain) and path prefix of site A
   - Check flags as well

Server Sets Cookies



- The first time a browser connects to a particular web server, it has no cookies for that web server
- When the web server responds, it includes a **Set-Cookie:** header that defines a cookie

Web Server Sets Cookie

The browser checks if the server may set the cookie, and if not, it will not accept the cookie.

domain: any domain-suffix of URL-hostname, except TLD

path: can be set to anything

[top-level domains, e.g. '.com']

example: host = "login.site.com"

allowed domains	disallowed domains
login.site.com	**user.site.com**
.site.com	**othersite.com**
	.com

⇒ **login.site.com** can set cookies for all of **.site.com** but not for another site or TLD

Problematic for sites like .berkeley.edu

Web Server Sets Cookie Example

Web server at foo.example.com wants to set cookie with domain:

domain	Whether it will be set, and if so, where it will be sent to
(value omitted)	*foo.example.com* (exact)
bar.foo.example.com	
foo.example.com	
baz.example.com	
example.com	
ample.com	
.com	

Credits: The Tangled Web: A Guide to Securing Modern Web Applications, by Michał Zalewski

Web Server Sets Cookie Example

Web server at foo.example.com wants to set cookie with domain:

domain	Whether it will be set, and if so, where it will be sent to
(value omitted)	*foo.example.com* (exact)
bar.foo.example.com *foo.example.com*	Cookie not set: domain more specific than origin **.foo.example.com*
baz.example.com	Cookie not set: domain mismatch
example.com *ample.com*	**.example.com* Cookie not set: domain mismatch
.com	Cookie not set: domain too broad, security risk

Credits: The Tangled Web: A Guide to Securing Modern Web Applications, by Michał Zalewski

Receiving Cookies

- A cookie can be accessed in mostly two ways:
  - When a user visits a site, the user's browser sends automatically relevant cookies
  - Javascript can access it via document.cookie

Browser Sends Cookie

Browser

Server

GET //URL-domain/URL-path

Cookie: NAME = VALUE

Goal: server only sees cookies in its scope

Browser sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain, and
- cookie-path is prefix of URL-path, and
- [protocol=HTTPS if cookie has "Secure" flag set]

Browser Sends Cookie Example

cookie 1	cookie 2
name = **userid**	name = **userid**
value = u1	value = u2
domain = **login.site.com**	domain = **.site.com**
path = **/**	path = **/**
non-secure	non-secure

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u1, userid=u2

http://othersite.com/ cookie: none

Browser Sends Cookie Example

<u>cookie 1</u>
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
non-secure

<u>cookie 2</u>
name = **userid**
value = u2
domain = **.site.com**
path = **/secret**
non-secure

http://checkout.site.com/secret/treasure	cookie: userid=u2
http://login.site.com/	cookie: userid=u1
http://othersite.com/secret	cookie: none

Browser Sends Cookie Example

cookie 1	cookie 2
name = **userid**	name = **userid**
value = u1	value = u2
domain = **login.site.com**	domain = **.site.com**
path = **/**	path = **/**
secure	non-secure

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u2

http**s**://login.site.com/ **cookie: userid=u1; userid=u2**

(arbitrary order)

Client Reads Cookie

- Setting a cookie in Javascript:

  document.cookie = “name=value; expires=…; ”

- Reading a cookie: alert(document.cookie)

  prints string containing all cookies available for document (based on [protocol], domain, path)

- Deleting a cookie:

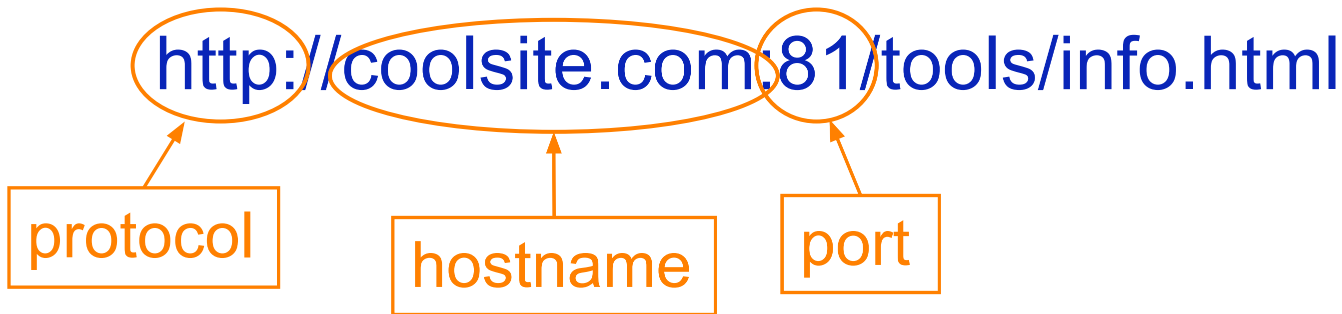
  document.cookie = “name=; expires= Thu, 01-Jan-70”

document.cookie often used to customize page in Javascript

Cookie Policy versus Same-Origin Policy

Recall: Same-Origin Policy

- Granularity of protection for same origin policy
- Origin = protocol + hostname + port



- Origin is determined by **string matching**! If these match, it is same origin, else it is not.

Cookie Policy vs SOP

- Consider Javascript on a page loaded from a URL U
- If a cookie is in scope for a URL U, it can be accessed by Javascript loaded on the page with URL U, unless the cookie has the httpOnly flag set

Cookie Policy vs SOP Example

<u>cookie 1</u>
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
non-secure

<u>cookie 2</u>
name = **userid**
value = u2
domain = **.site.com**
path = **/**
non-secure
Http-Only

http://checkout.site.com/	cookie: none
http://login.site.com/	cookie: userid=u1
http://othersite.com/	cookie: none

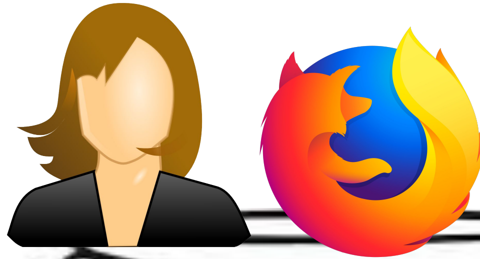
JS on each of these URLs can access all cookies that would be sent for that URL if the httpOnly flag is not set

Indirectly Bypassing SOP using Cookie Policy

- Since the cookie policy and the same-origin policy are different,
  - there are corner cases when one can use cookie policy to bypass same-origin policy

Indirectly Bypassing SOP using Cookie Policy

Victim user browser



financial.example.com
web server

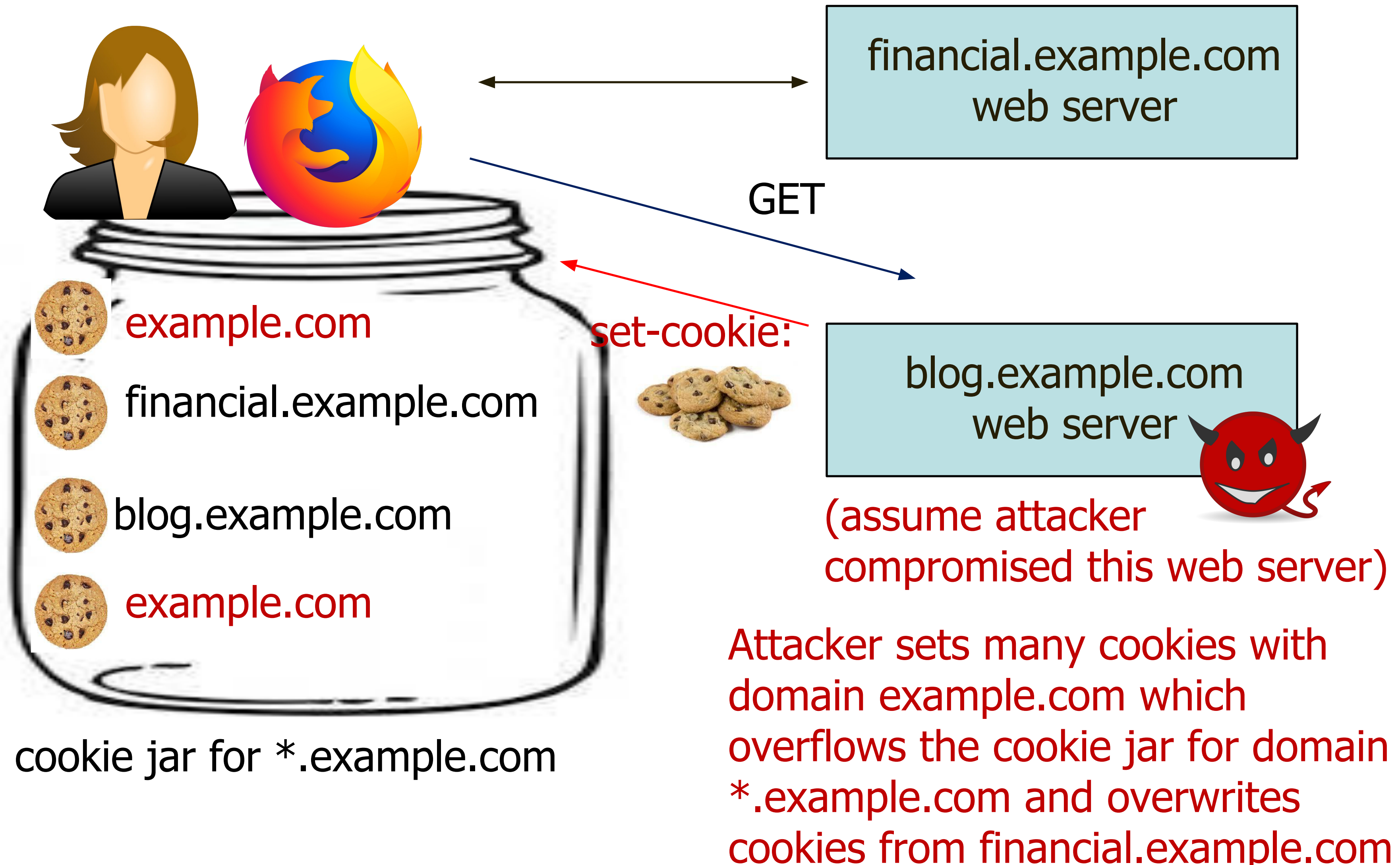
blog.example.com
web server

(assume attacker compromised this web server)

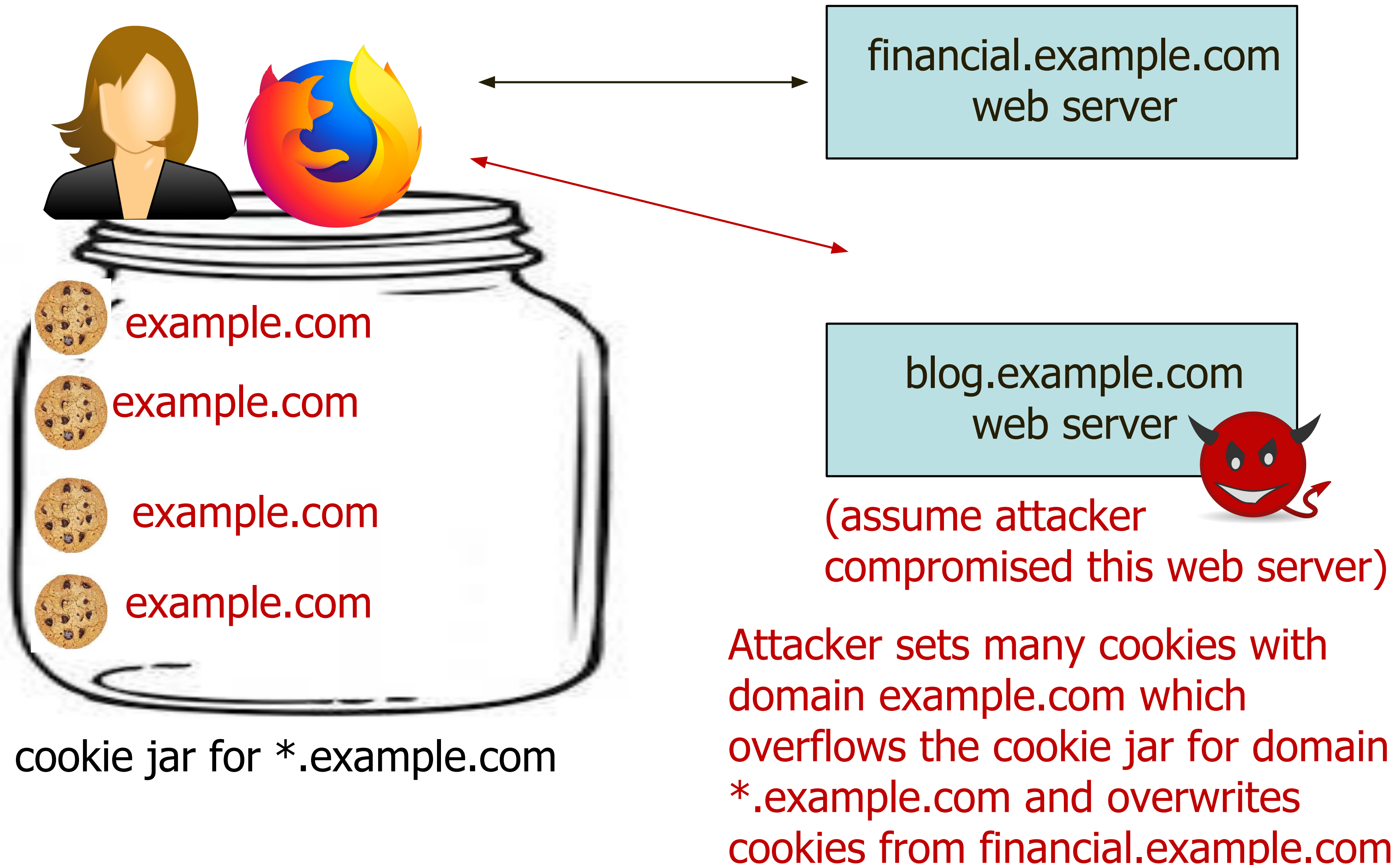
cookie jar for *.example.com

Browsers maintain a separate cookie jar per domain group, such as one jar for *.example.com to avoid one domain filling up the jar and affecting another domain. Each browser decides at what granularity to group domains.

Indirectly Bypassing SOP using Cookie Policy



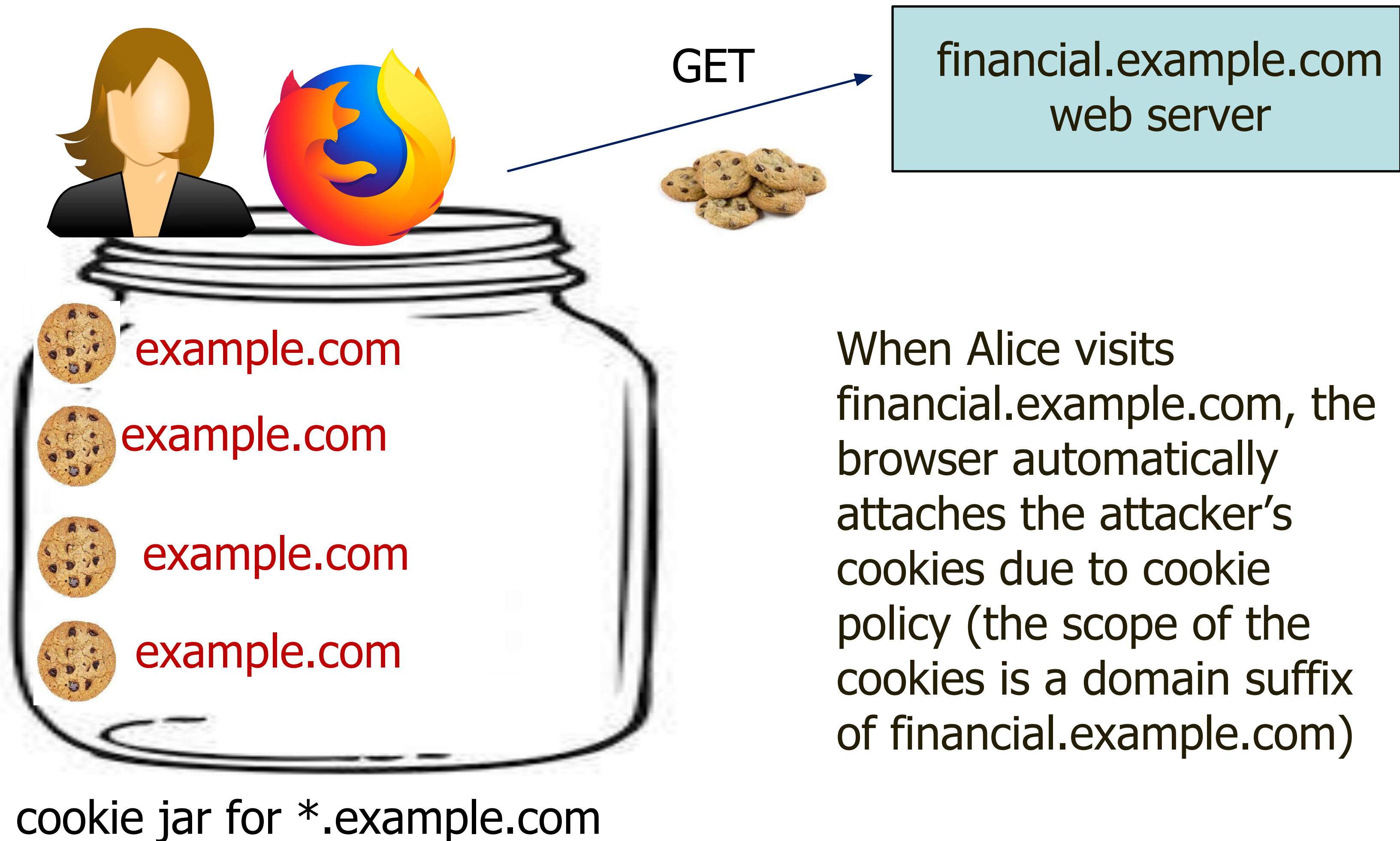
Indirectly Bypassing SOP using Cookie Policy



Attacker sets many cookies with domain example.com which overflows the cookie jar for domain *.example.com and overwrites cookies from financial.example.com

Indirectly Bypassing SOP using Cookie Policy

Victim user browser



When Alice visits financial.example.com, the browser automatically attaches the attacker’s cookies due to cookie policy (the scope of the cookies is a domain suffix of financial.example.com)

Why is this a problem?

Indirectly Bypassing SOP using Cookie Policy

- Victim thus can login into attackers account at financial.example.com
- This is a problem because the victim might think its their account and might provide sensitive information
- This bypassed same-origin policy (indirectly) because blog.example.com influenced financial.example.com

RFC6265

- For further details on cookies, checkout the standard RFC6265 “HTTP State Management Mechanism”

  https://tools.ietf.org/html/rfc6265

- Browsers are expected to implement this reference, and any differences are browser specific

Break Time: Ruta Jawale

- Got stuck under the Swiss Alps

Session Management

Sessions

- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (Gmail - two weeks) or short
  - without session management:

    users would have to constantly re-authenticate

- Session management:
  - Authorize user once;
  - All subsequent requests are tied to user

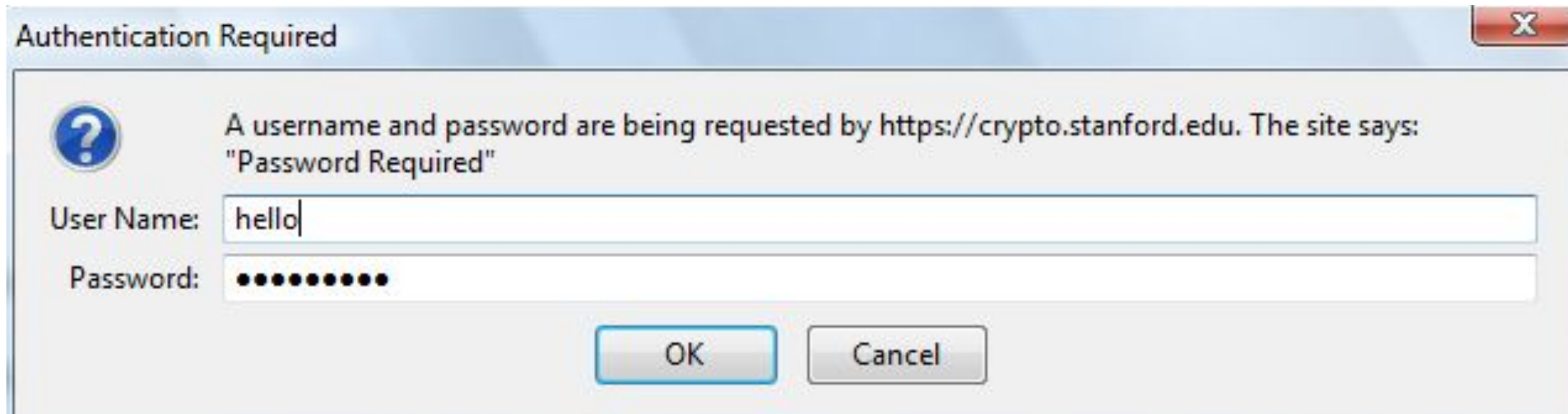
Historical: HTTP Authentication

One username and password for a group of users

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"



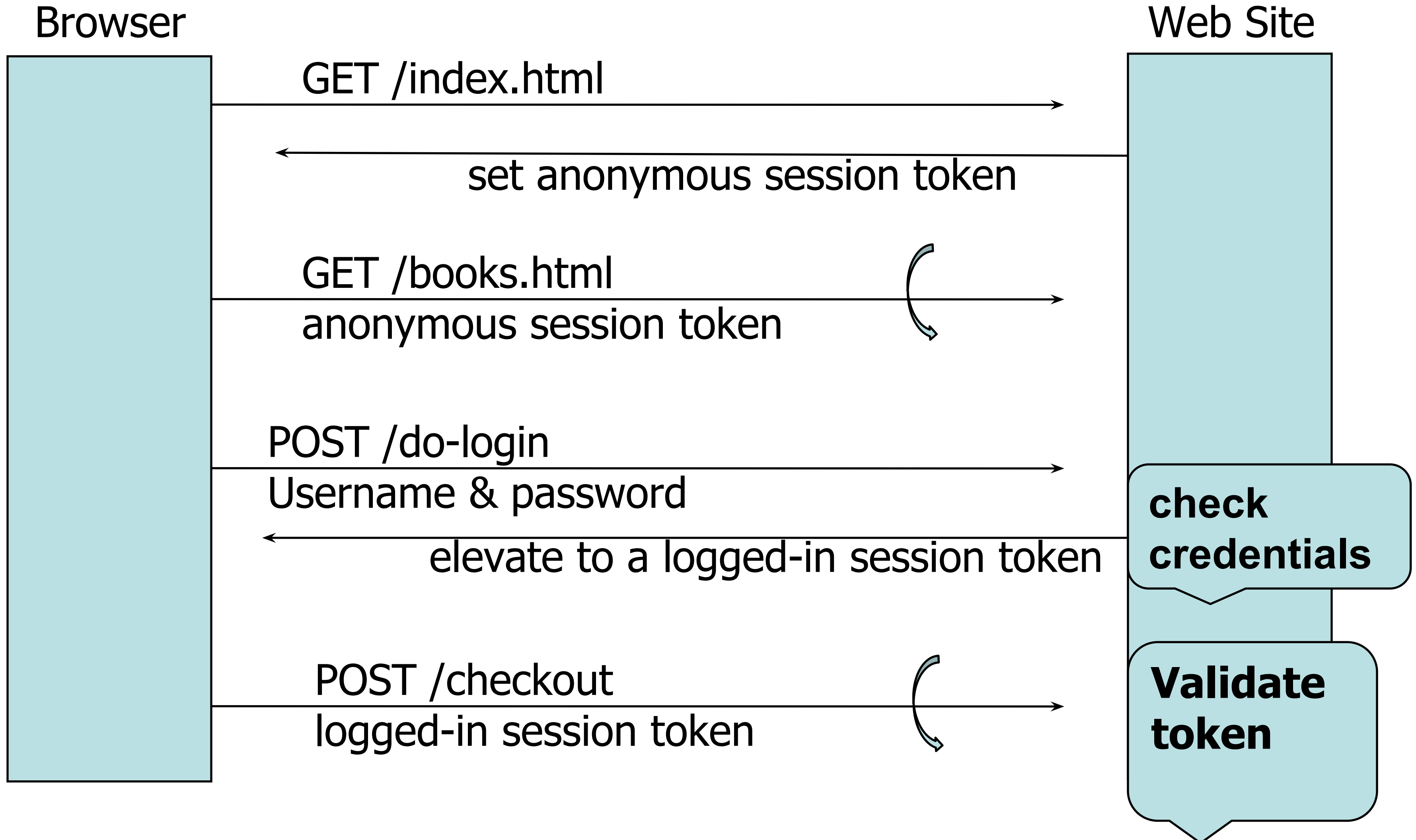
Browsers sends hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdfgkjheczI1NXRleHQ=

HTTP Authentication Problems

- Hardly used in commercial sites
  - User cannot log out other than by closing browser
    - What if user has multiple accounts?
    - What if multiple users on same computer?
  - Site cannot customize password dialog
  - Confusing dialog to users
  - Easily spoofed

Session Tokens



Storing Session Tokens

- Browser cookie:

  Set-Cookie: SessionToken=fduhye63sfdb

- Embed in all URL links:

  https://site.com/checkout ? SessionToken=kh7y3b

- In a hidden form field:

  <input type="hidden" name="sessionid" value="kh7y3b">

Storing Session Tokens

- Browser cookie:

  browser sends cookie with every request,
  even when it should not (CSRF)

- Embed in all URL links:

  token leaks via HTTP Referer header

  users might share URLs

- In a hidden form field: short sessions only

Better answer: a combination of all of the above (e.g., browser cookie with CSRF protection using form secret tokens)

Cross-Site Request Forgery (CSRF)

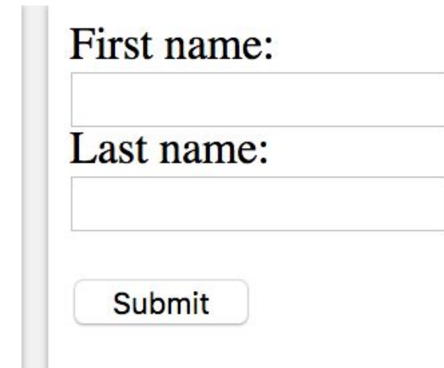
Top 10 web vulnerabilities

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

HTML Forms

- Allow a user to provide some data which gets sent with an HTTP POST request to a server

```
<form action="bank.com/action.php">
First name:  <input type="text"
               name="firstname">
Last name:<input type="text"
            name="lastname">
<input type="submit"
   value="Submit"></form>
```



When filling in Alice and Smith, and clicking submit, the browser issues

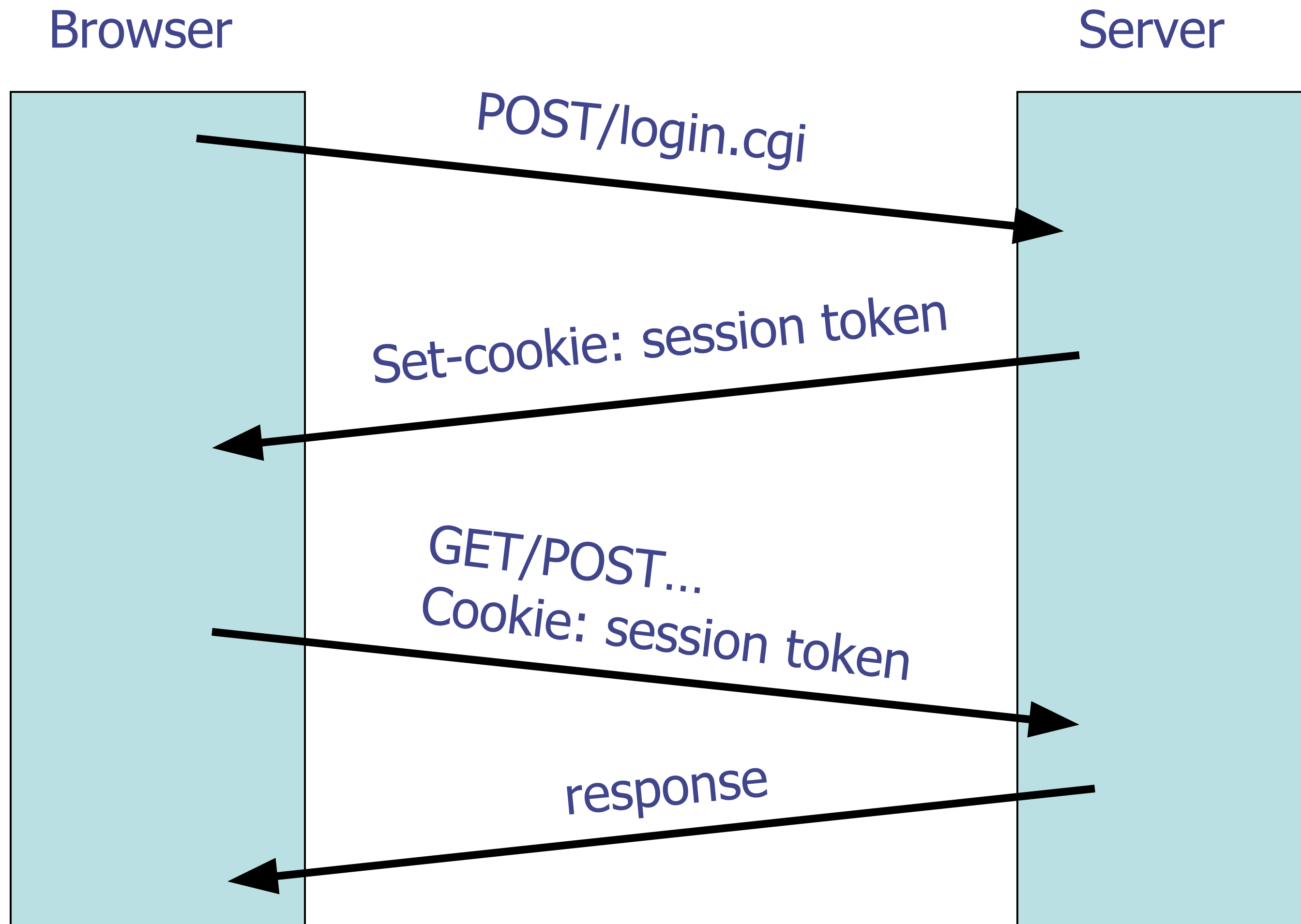
HTTP POST request bank.com/action.php?firstname=Alice&lastname=Smith

As always, the browser attaches relevant cookies

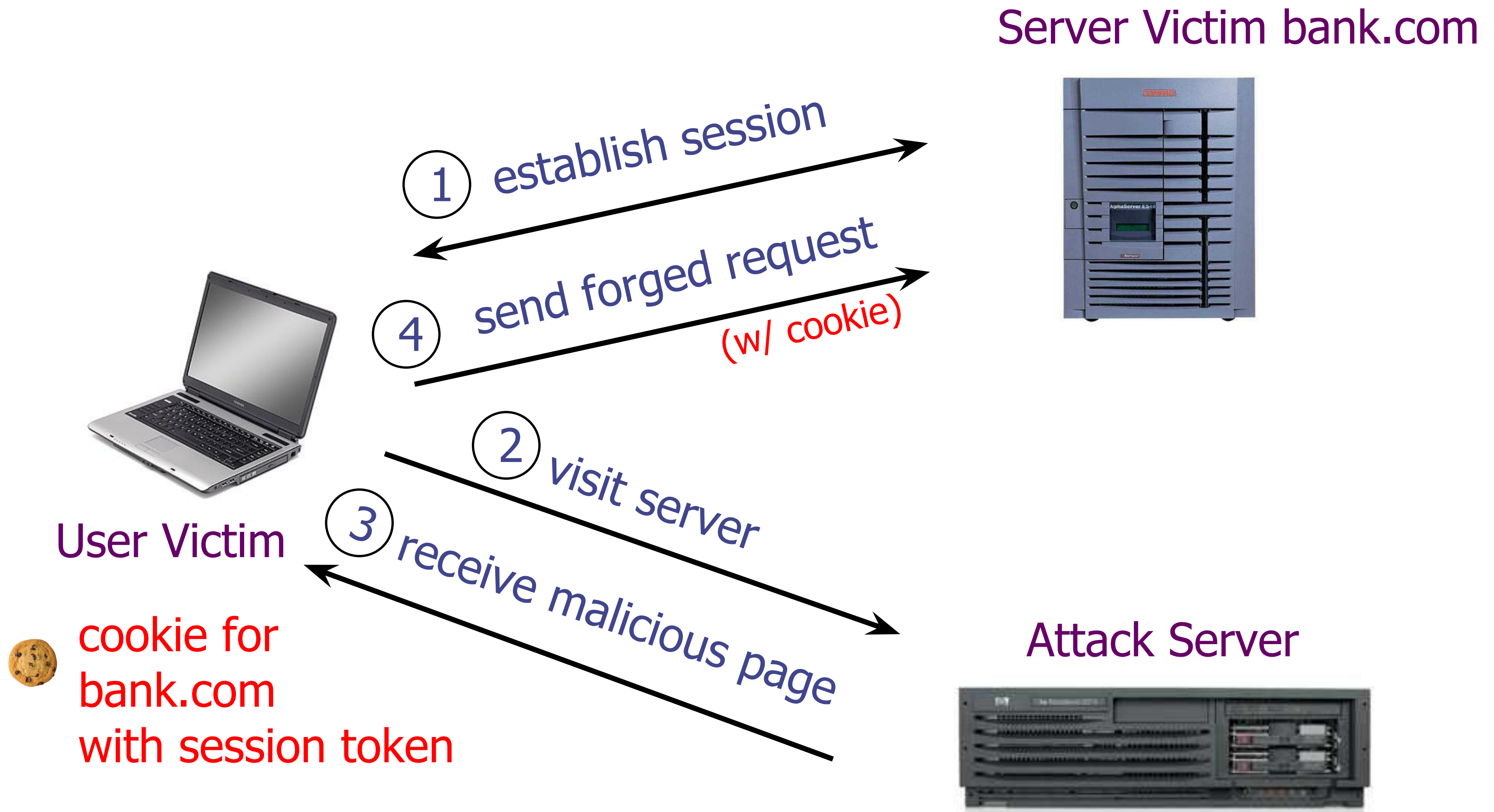
Consider: Cookie Stores Session Token

- Server assigns a session token to each user after they logged in, places it in the cookie
- The server keeps a table of username to current session token, so when it sees the session token it knows which user

Cookie Stores Session Token



Cross-Site Request Forgery (CSRF)



What can go bad?

URL contains transaction action

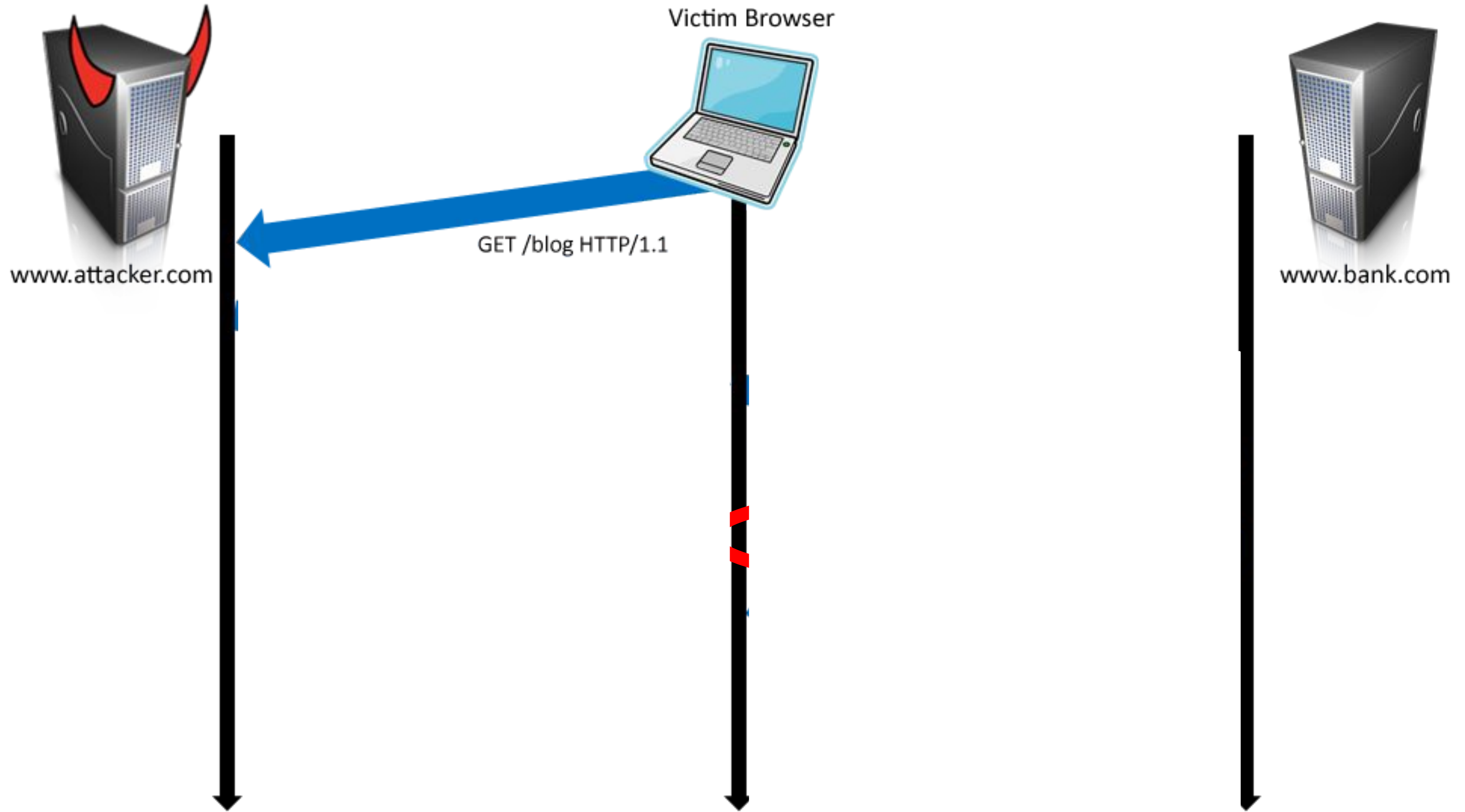
Cross-Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits malicious site containing:

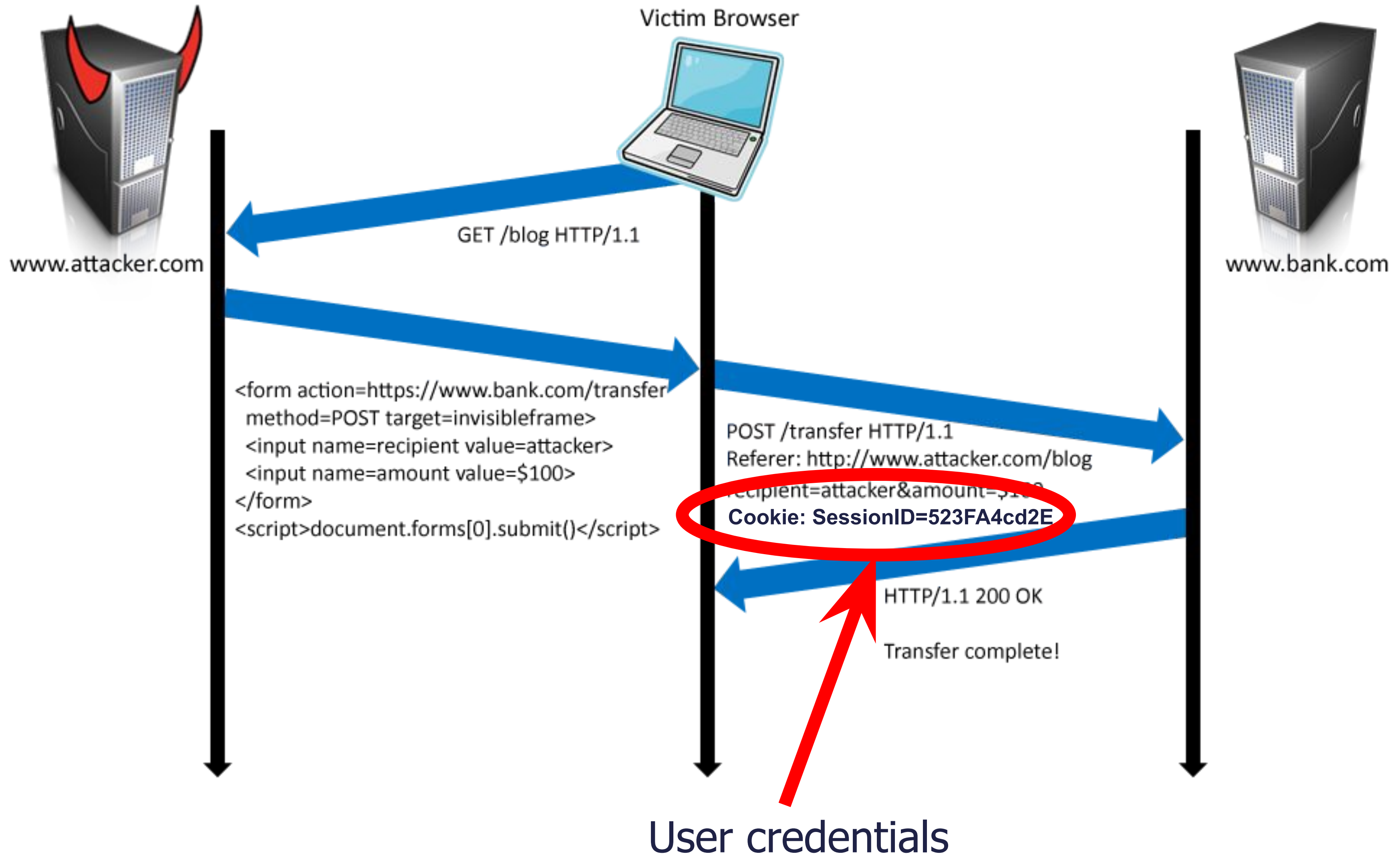
```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF)



Demo

YouTube 2008 CSRF attack

An attacker could
- add videos to a user’s "Favorites,"
- add himself to a user’s "Friend" or "Family" list,
- send arbitrary messages on the user’s behalf,
- flagged videos as inappropriate,
- automatically shared a video with a user’s contacts, subscribed a user to a "channel" (a set of videos published by one person or group), and
- added videos to a user’s "QuickList" (a list of videos a user intends to watch at a later point).

Home → Security → Facebook Hit by Cross-Site Request Forgery Attack

Facebook Hit by Cross-Site Request Forgery Attack

By Sean Michael Kerner / August 20, 2009

Angela Moscaritolo

September 30, 2008

Popular websites fall victim to CSRF exploits

CSRF Defense

CSRF Defense

- CSRF token

```
<input type=hidden value=23a3af01b>
```

- Referer Validation

```
Referer: http://www.facebook.com/home.php
```

- Origin Header Validation
  - See discussion
- Others (e.g., custom HTTP Header)

CSRF Token

1. goodsite.com server wants to protect itself, so it includes a secret token into the webpage (e.g., in forms as a hidden field)
2. Requests to goodsite.com include the secret
3. goodsite.com server checks that the token embedded in the webpage is the expected one; reject request if not

Can the token be?

- 123456
- Dateofbirth

No, CSRF token must be hard to guess by the attacker

CSRF Token

- The server stores state that binds the user's CSRF token to the user's session id
- Embeds CSRF token in every form
- On every request the server validates that the supplied CSRF token is associated with the user's session id
- Disadvantage is that the server needs to maintain a large state table to validate the tokens.

Referer Validation

- When the browser issues an HTTP request, it includes a referer header that indicates which URL initiated the request
  - Referer header could be used to distinguish between same site request and cross site request

Referer Validation

Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

Remember me

Login or Sign up for Facebook

Forgot your password?

Referer Validation

- HTTP Referer header
  - Referer: http://www.facebook.com/
  - Referer: http://www.attacker.com/evil.html
  - Referer:
    - Strict policy disallows (secure, less usable)
    - Lenient policy allows (less secure, more usable)

Privacy Issue: Referer Validation

Privacy Issues with Referer header:

- The referer contains sensitive information that impinges on the privacy
- The referer header reveals contents of the search query that lead to visit a website.
- Some organizations are concerned that confidential information about their corporate intranet might leak to external websites via Referer header

Privacy Issue: Referer Validation

- Referer may leak privacy-sensitive information

  ```
  http://intranet.corp.apple.com/
  projects/iphone/competitors.html
  ```

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser

Summary

- Cookies add state to HTTP
  - Cookies are used for session management
  - They are attached by the browser automatically to HTTP requests
- CSRF attacks execute request on benign site because cookie is sent automatically
- Defenses for CSRF:
  - embed unpredicatable token and check it later
  - check referer header